Optimization of YARN Hierarchical Resource Scheduling Algorithm

Hengyi Wei*, Dengxin Luo and Lei Liang

The School of Electronics and Information Engineering, Xi'an Jiaotong University, No.28, Xianning West Road, Xi'an, 710049 Shanxi, China

ABSTRACT

We propose a hierarchical resource scheduling algorithm to solve the YARN resource problem, YARN is the resource scheduling management platform of Hadoop. Based on the resource scheduling DRF algorithm, the algorithm improves the fairness of the hierarchical resource scheduling by adjusting the calculation method of the usage rate of the parent node's primary resource. This paper establishes a hierarchical resource preemption model based on the characteristics of YARN task, and designs a framework to dynamically determine the preemptive task. The algorithm improves cluster resource's utilization and rationality. The experimental results show our work can effectively shorten the overall run time and average YARN platform response time, improve the usage rate of the distributed cluster resources, and improve the Fair scheduling capability of YARN under the multiple queues and multiple resources.

INTRODUCTION

Hadoop platform is a distributing computing platform in big data and cloud computing environment. Developers can use it to store and analyze massive data. Hadoop's high reliability and scalability to make it quickly become mainstream distributed computing platform. In this paper, the fairness and rationality of resource scheduler of the Fair scheduler in the YARN [1] platform of Hadoop2.X are studied, and the corresponding resource scheduling improvement method is proposed to improve the scheduling ability of YARN effectively.

In this paper, we discuss resource scheduling fairness and resource preemption rationality in YARN. In view of the shortcomings of resource scheduling proposed. On the basis of hierarchical scheduling, we improve the fairness problem in resource scheduling by adjusting the calculation method of the primary resource usage rate of queue nodes in DRF algorithm. According to the characteristics of the task of YARN, we propose an algorithm that dynamically determines the preemptive task to improve the performance of resource preemption.

This paper designs a YARN resource scheduler (DMHB Scheduler) to verify the fairness and rationality of the improved resource scheduling algorithm. Then the DMHB scheduler is tested in an actual cluster environment. The results show that the scheduler can effectively shorten the overall run time and average response time in the YARN platform, improve utilization rate of cluster resources, and improve the ability of YARN to be scheduled evenly under the hierarchical resource scheduling.
CURRENT RESEARCH

In YARN, resource scheduling [2] is the stripping and reconstruction of job scheduling in Map-Reduce. For the dispatching method of Map-Reduce, the relevant scholars have done many relative works. Perret [3] based on the deadline of the schedule to fully consider the relationship between tasks and data locality. Kai Wang [4] et al. extracted information from the historical execution record of the job to estimate the remaining completion time, but this way only consider Map-Reduce type of job. Zaharia [5] proposed resource-based delay scheduling, by improving Hadoop's data local to improve the performance of Yarn's Fair scheduler. The literature [6] proposed throughput-based scheduler to improve the performance on heterogeneous Hadoop nodes. The literature [7-9] studied the problem of data localization, map and reduce task independence and scheduling optimization through delay scheduling in a shared cluster. In the DRF [10] algorithm proposed by the open source management platform (Mesos [11]), the primary resource is the largest resource of the resource required for the task. In the hierarchical queue tree, it has the parent queue node, the child queue node and the leaf nodes, which all belong to the queue node. However, the queue node also contains the application.

DISADVANTAGE OF FAIR SCHEDULER

Currently, three schedulers can be configured in YARN, namely FIFO scheduler, Capacity scheduler and Fair scheduler. Only the Capacity scheduler and the Fair scheduler support DRF resource scheduling algorithms and resource preemption between queues. Compared with the Fair scheduler and the Capacity scheduler, the Fair scheduler contains a variety of flexible configuration strategies. Therefore, it is generally accepted that the Fair scheduler contains all the functions of the Capacity scheduler. The research work of this paper is also based on the DRF algorithm in the Fair scheduler. Although the presence of queues in YARN facilitates the management of cluster resources, they still have two shortcomings:

1) Defects in hierarchical resource scheduling: During the resource scheduling process, some queues within the hierarchical queue tree are prone to "starve" the phenomenon. Figure 1 shows the cluster resource queue hierarchical structure, the total cluster resource is 1GB of memory and 2 virtual CPUs. <x, y> indicates that the queue node occupies x GB of memory and y virtual CPUs.

The DRF algorithm tries to maximize the minimum amount of resources in all primary resources in the cluster and uses the ratio of the primary resource occupied by the queue node to the total amount of the cluster's primary resource to represent the primary resource usage rate of the queue node. In Figure 1, leaf nodes $n_{1,1}, n_{2,1}, n_{2,2}$ of the primary resource usage rates are 50%, 50%, 100% respectively.

277
When an application in the leaf node $\text{n}_{2,1}$ is completed, the resources occupied by this application will be released. At this time, the scheduling policy under the DRF will trigger the resource scheduling because the cluster has free resources. But the primary resource usage rate of the queue node $\text{n}_2$ is 100%, then this means that the released resource will be accepted by the queue node $\text{n}_1$, and the leaf node $\text{n}_{1,1}$ will get the resource. If the leaf node $\text{n}_{2,1}$ continues to release the resource, then the process of resource allocation will continue to cycle, finally the leaf node $\text{n}_{2,1}$ will appear resources "starved" phenomenon, leading to the queue tree internal queue node resource scheduling unfair. In terms of shortcomings we discussed above, this paper proposes to adjust the calculation method of the usage rate of the primary resource to solve.

2) Defects in resource preemption: In the fair scheduling, resource preemption takes place in this situation, the resource usage rate of the cluster is high, a new job requests resources in the idle queue node, and the resources of the queue nodes in the cluster can be shared. Unreasonable resource preemption will reduce the resource usage rate of the cluster. The unreasonable phenomenon of resource preemption includes the problem of sorting time is too long, blind preemption, preemption is based on a single, ignore the cost of killing other tasks and other issues. When sort time is too long, the scheduler needs to prioritize the tasks that are running in the cluster. If the cluster is large and the number of tasks is large, the preemption time will be too long and the heartbeat information of the cluster queue node cannot get immediately response, resulting in reduced scheduling efficiency. Blind preemption is due to a variety of resources under the preemption can not be in accordance with the actual needs of the resource type, such as: a queue node requires a memory resource, but it seize the CPU resource. To seize the single is because of the preemption can only be sorted according to the priority of the task, and the actual priority can only be static response to the task of scheduling resources, to not take into account the dynamic factors, such as: running time, the localization of the task factors. Ignoring the cost of killing the task is to seize the total amount of resources needed to get, but not to consider the importance of killing the task and to restore its price. In this paper, we improve the resource preemption model in the fourth step through the establishment of the hierarchical resource preemption model. First, we select the appropriate queue node. Then we select the appropriate application from the queue node. Finally we select the appropriate container from the application. In the process, in order to achieve the purpose of reasonable resource preemption, we need to consider the locality of the data, the completion of the request, the weight of the remaining time.
ALGORITHM DESIGN

In terms of the problems existing in Fair scheduler in current hierarchical resource scheduling mode, this paper designs the DMHB scheduler according to scheduler interface reserved in YARN.

Improvement of Hierarchical Resource Scheduling

The DRF algorithm in Fair scheduler does not appear resource "starvation" phenomenon in non-hierarchical resource scheduling, but the hierarchical resource scheduling can appear resources "starvation" phenomenon. The main reason is current queue tree does not take into account the balance of usage rate of the primary resource in queue node, that is the fairness of the DRF algorithm. This paper improves the method of calculating the primary resource usage rate of queue nodes in the queue tree. An improved resource scheduling algorithm is given below.

1) Adjust the primary resource usage rate of queue node. Recalculate the primary resource usage rate of each queue node in current hierarchical queue tree. As follows:
   (1) the least value \( M = \min \) of the primary resource usage is selected from all child queue nodes of parent queue node of the current queue node;
   (2) re-adjust the resource availability of all child queue nodes of parent queue node to \( M \);
   (3) the resource availability of all sub-queue nodes is added to resource availability of the parent queue node, and the primary resource usage rate of parent queue node is obtained accordingly.

2) remove the block node in resource scheduling. It is necessary to remove the nodes (applications) that are block to resource scheduling in the current queue tree, and reduce the impact on the primary resource usage rate of parent queue nodes while recalculating the primary resource usage rate of queue nodes in hierarchical queue tree. The following is the definition of an block node:
   (1) If the resource requested by an application in leaf node has been allocated, this application will be considered to be blocked.
   (2) If all applications of a queue node are block, this queue node will be considered to be blocked.
   (3) If all leaf nodes (applications) of a parent queue node are block, this parent queue node is also considered to be blocked.

The improved primary resource usage rate of parent queue node is calculated as follows:

\[
drf_{\text{new}}(\text{parent}) = \text{drf}((\sum_{i=1}^{n} \frac{M}{D_i} \times \text{resource}_i) \times \text{drf}_{\text{old}}(\text{parent})) \tag{1}
\]

\[
M = \min(D_1, D_2, ..., D_i) \tag{2}
\]

\[
D_i = \text{drf}(n_i) \tag{3}
\]

In equation 1, \( \sum_{i=1}^{n} \left( \frac{M}{D_i} \times \text{resource}_i \right) \) is ability of the parent queue node demands resources; \( \text{drf}_{\text{old}}(\text{parent}) \) is the primary resource usage rate of parent queue node when no new resource is scheduled. In equation 2, \( M \) is the least primary resource usage
rate in all sub-queue nodes of parent queue node; in equation 3, $n_i$ is a queue node; $D_i$ is the primary resource usage rate of queue node $n_i$; $drf()$ is used to calculate the primary resource usage rate of queue node; $resource$ is the resource usage rate of queue node $n_i$.

**Improvements to Resource Preemption Methods**

In order to solve these problems of Sort time is too long, blind preemption, preemption is based on a single, ignore the cost of killing other tasks, this paper improves the resource preemption stratification model in hierarchical resource scheduling mode. Resource preemption is done in units of queues, different applications need to be submitted to the leaf nodes to be executed. In this paper, the resource preemptive model is designed to select appropriate queue node, and then select appropriate application from the queue node. Finally, select appropriate container from the application. The container is an abstract representation of resource in YARN. A container runs a task. Figure 2 shows a hierarchical model of resource preemption.

![Stratified model of resource preemption](image_url)

**Figure 2. Stratified model of resource preemption.**

1) Select the preempted target queue node based on preemptive appropriateness

$$F_i = \sum_{p \in (1,k)} R_p \cdot \Delta[i, p] \cdot \omega_p$$  \hspace{1cm} (4)

In equation 4, $F_i$ is suitability of current application for the preempted resource R; $i$ is queue node number; $k$ is resource type of cluster scheduling; $R_p$ is amount of p-type resource in the resource R that needs to be preempted in cluster;p is a kind of resource; $\Delta[i, p]$ is amount of resources available for queue node $i$ for resource $p$; $\omega_p$ is the weight of resource $p$; A queue node of larger appropriateness $F_i$ will serve as the target queue node to be preempted.

2) Select the preempted application according to appropriateness of preempting resource R

$$F_i = \sum_{p \in (1,k)} R_p \cdot A[i, p] \cdot \omega_p$$  \hspace{1cm} (5)

In equation 5, $F_i$ is suitability of current application for the preempted resource R;
\( R_p \) is value of p-type resource in the resource R that needs to be preempted in cluster; p is a resource; \( A[i, p] \) is amount of resources available for application i for resource p; \( \omega_p \) is weight of resource p. A application of larger appropriateness \( F_i \) will serve as the preemptive target.

3) Three factors of choose to be preempted task container

1) Data locality weight

In resources preemption, in order to ensure the application of computing resources in the data of local, to reduce the loss of the task has been run. When determining container to be preempted, the container that is already in the data locality will get a higher weight to ensure data localization as much as possible. Here we use the equation 6 to represent the weight of data localization in the decision container (weight_{locality}).

\[
weight_{locality} = \begin{cases} 
1 & \text{locality} = \text{true} \\
0 & \text{locality} = \text{false}
\end{cases} 
\]  
\( (6) \)

In equation 6, the locality indicates whether the data locality of the task is satisfied.

2) Request priority weight

Here, the priority refers to priority of resource request reserved in container. Each container is allocated by the request of the applied resource. For example, the priority of Map Task is 20, the priority of Reduce Task is 10, and the priority of MrAppMaster (AM) is 1 (The smaller the value of task's priority, the higher its priority). Here, all the containers in application are normalized according to priority factors, and the priority is determined in determining the weight of container. The normalized subclass used is as follows:

\[
weight_{priority} = \begin{cases} 
\frac{1}{P_{max} - P_i} & P_{max} = P_{min} \\
\frac{P_{max} - P_{min}}{P_{max} - P_i} & P_{max} \neq P_{min}
\end{cases} 
\]  
\( (7) \)

\[
P_{max} = \max(p_1, p_2, p_3, \ldots, p_n)
\]  
\( (8) \)

\[
P_{min} = \min(p_1, p_2, p_3, \ldots, p_n)
\]  
\( (9) \)

In equation 7, weight_{priority} is weight of priority factor normalized; \( p_i \) is weight of resource request in container; n is number of all container collections.

3) The weight of remaining completion time

An application may contain multiple priority requests, each request will correspond to the dispatcher assigned container. The application runs several independent tasks (in container), each container contains the requested amount of resources. If there is already a part of container in this application because of the task is completed (only the successful completion of task, regardless of the task of performing the failure and the task of being killed) and release resources, here we will be based on this part of task execution time estimate the remaining time of execution of other tasks in application, giving the corresponding time weights. The following formula is given:

\[
speed_{i,p} = \frac{container[i, p]}{time_i} \quad p \in (1,k), i \in (1,n)
\]  
\( (10) \)
speed_p = \frac{n}{\sum_{i=1}^{k} \text{speed}_{i,p}} \quad p \in (1, k) \quad (11)

time_{j,estimate} = \max(\frac{\text{container}[j,1]}{\text{speed}_p}, \ldots, \frac{\text{container}[j,p]}{\text{speed}_p}) \quad j \in (1, m), p \in (1, k) \quad (12)

weight_{j, time} = \frac{\text{time}_{j, used}}{\text{time}_{j, estimate}} \quad j \in (1, m) \quad (13)

In the above four equations, \(i\) is the subscript that the container has been successfully run in application; \(\text{time}_i\) is the execution time of \(\text{container}_i\); \(\text{speed}_{i,p}\) is the resource consumption rate of resource \(p\) in the successful implementation of \(\text{container}_i\); \(\text{speed}_p\) is the historical average value of \(p\)-type resource consumption rate; \(\text{container}_j\) is a \(\text{container}\) that is running in application; \(m\) is the number of containers that are running in application; \(n\) is the number of containers that have been completed in application; \(\text{time}_{j, estimate}\) is the \(\text{container}_j\) theoretical run duration value obtained from the historical resource consumption rate; \(\text{time}_{j, used}\) is the difference that \(\text{container}_j\) from the start of the scheduled request to the current execution time. According to the above resource rate estimation method, we can get the ratio of \(\text{container}_j\) for current running time.

Here, the weight of task \(i\) (container) in operation is determined based on the weighted values given by the above three factors.

\[\text{weight}_i = \alpha \times \text{weight}_{i, locality} + \beta \times \text{weight}_{i, priority} + \gamma \times \text{weight}_{i, time} \quad (14)\]

In equation 14, \(\text{weight}_{i, locality}\) is the weight of the resource data locality of \(\text{container}_i\); \(\text{weight}_{i, priority}\) is the priority weight of \(\text{container}_i\); \(\text{weight}_{i, time}\) is the time estimation weight of the running \(\text{container}_i\); \(\alpha, \beta, \gamma\), respectively, is the locality, priority, run-time estimated weight. The relationship between the three variables is: \(\alpha + \beta + \gamma = 1\). The smaller the weight value of \(\text{weight}_i\), indicates that the task (container) is preempted.

**EXPERIMENTAL RESULTS AND ANALYSIS**

**Experimental Design and Environment**

In experiment, all hosts are same configuration and are located in same rack Resource-Manager 2 (active and standby), Node-Manager 40. The operating system uses 64-bit Red Hat 4.4.6-3, Hadoop uses Hadoop-2.3.0-cdh5.0.0. Queue and application type settings are as follows.

Here, the representation of resource is <memory, CPU>, the unit of memory is GB. The hierarchical organization of the queue tree is shown in Figure 3. In order to more accurately reflect the performance of improved scheduler, 100 load applications will be constructed. The composition of the specific application is shown in TABLE I.
Figure 3. Performance testing of hierarchical queue tree structure.

<table>
<thead>
<tr>
<th>The number of tasks included in application</th>
<th>Number of load applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–30</td>
<td>75</td>
</tr>
<tr>
<td>30–150</td>
<td>9</td>
</tr>
<tr>
<td>150–499</td>
<td>6</td>
</tr>
<tr>
<td>500–1000</td>
<td>10</td>
</tr>
</tbody>
</table>

All sample applications will only use both memory and CPU resources. The application is divided into memory type application and CPU type application. The resource type of the application request in the application includes <1.5G, 1vCores>, <1G, 1vCores> and <1G, 1.5vCores>. Where the leaf nodes $n_{1,1}$ will only submit tasks of type <1G, 1.5vCores>, and the execution time of these tasks is relatively long. Applications of 500 to 1000 tasks will be submitted to leaf nodes of $n_{1,1}$, $n_{1,2}$, $n_{2,1}$, and other types of applications will be submitted to the $n_2$ queue evenly. The application in each leaf node is submitted in order, and the order of the two schedulers for performance comparison is the same.

Results and Analysis

In order to analyze the resource scheduling process and the method of resource preemption in multiple queues, a number of applications will be built to verify the performance of the scheduler.

(1) Comparison of throughput of leaf nodes in cluster

<table>
<thead>
<tr>
<th>scheduler</th>
<th>Leaf Node Task Number Throughput / Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMHB</td>
<td>$n_{1,1}$</td>
</tr>
<tr>
<td></td>
<td>33</td>
</tr>
<tr>
<td>Fair(DRF)</td>
<td>18</td>
</tr>
</tbody>
</table>

TABLE II shows the comparison of leaf node throughput rate for load applications. Compared with the Fair scheduler, the improved scheduler DMHB makes throughput rate of leaf nodes $n_{1,1}$, $n_{1,2}$ and $n_{2,1}$ more balanced and the gap is
significantly reduced. The throughput rate of leaf nodes $n_{1,1}$ and $n_{1,2}$ is larger than that of other leaf nodes, while the throughput rate of other leaf nodes is not much different. The reason is that the leaf node $n_{1,1}$ is a CPU type application, and the task runs longer, while other queue nodes have a relatively short run time. Under this condition, the application of each queue node in the order of submission will result in resource preemption between the queue nodes and the availability of resources of the internal queue node $n_{1,2}$ can not be adequately guaranteed. The DMHB scheduler can reduce the resource availability of the internal queue node and ensure the resource usage rate of queue node $n_{1,2}$ in queue node $n_{1,1}$, and improve resource usage rate. However, in Fair scheduler, the queue node $n_{1,2}$ frequently preempts the resources of $n_{1,1}$, because it does not get minimum amount of resources, but this part of the resource will be acquired by the $n_2$ and $n_3$. Unbalanced resource scheduling results in lower throughput rates of $n_{1,1}$ and $n_{1,2}$.

(2) The response time of each queue node of the cluster

Here, the response time of the leaf node is defined as the duration time of application execution in queue node. Table 3 shows the response time of leaf nodes using Fair scheduler and DMHB scheduler.

<table>
<thead>
<tr>
<th>scheduler</th>
<th>Leaf node application response time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n_{1,1}$</td>
</tr>
<tr>
<td>DMHB</td>
<td>87</td>
</tr>
<tr>
<td>Fair(DRF)</td>
<td>35</td>
</tr>
</tbody>
</table>

TABLE III shows the comparison of the response time of the leaf nodes in the load application, that is the execution duration time comparison in the queue node. It can be seen that the improved scheduler DMHB compares the queue nodes $n_{1,1}$, $n_{1,2}$ and $n_{2,1}$ application execution duration time is more balanced, the gap is significantly reduced.

Two experiments show that the improved resource scheduler DMHB makes the throughput rate of leaf nodes more balanced than the Fair scheduler, and "starvation" phenomenon does not occur again, and duration time of the application between the leaf nodes also more balanced, the gap was significantly reduced, greatly improving the resource usage rate in cluster.

CONCLUSIONS

This paper improves the resource scheduling algorithm used in current YARN. This paper analyzes the phenomenon of "starvation" in application of resource scheduler in the hierarchical resource scheduling process, and proposes an improved primary resource scheduling algorithm. By recalculating the primary resource usage
rate of queue node, the "starvation" phenomenon of queue node is improved in resource scheduling, and it improves the fairness of the hierarchical resource scheduling. In addition, the hierarchical model of resource preemption is proposed, which improves the way of resources preemption in hierarchical resources. This paper analyzes the problem of resource preemption between dispatching queues and discusses the irrationality of current resource preemption. And the paper proposes a algorithm to determine the preemptive task in the combination weighting application, which improves the rationality of resource preemption and improves the resource usage rate of the cluster. In summary, the improved algorithm in this paper can improve the resource utilization rate of the cluster and reduce the starvation of the queue nodes, and ensure the fairness of hierarchical resource scheduling and the rationality of resource preemption in YARN.

REFERENCES